Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «МиАПР»

Тема: «Нелинейные ИНС в задачах прогнозирования.»

Выполнил:

Студент 2 курса

Группы ПО-7(2)

Фурсевич Д.С.

Проверил:

Крощенко А.А.

2021

**Цель работы:** изучить обучение и функционирование нелинейной ИНС при решении задач распознавания образов.

**Ход работы**

Вариант 11

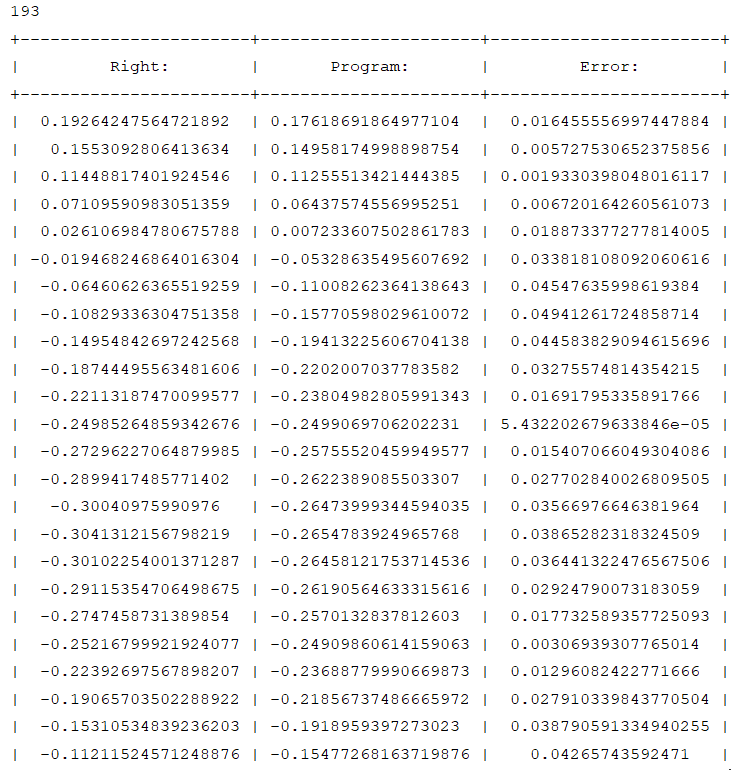
**Задание:**

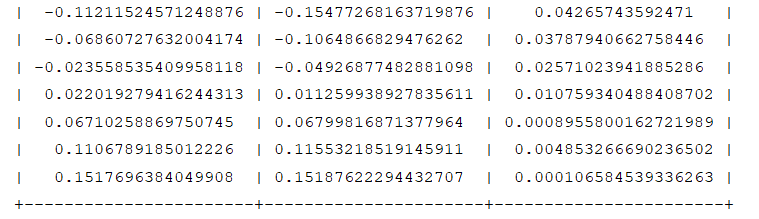
Спрогнозировать нелинейный временной ряд, применяя параметры лабораторной работы №3. При этом необходимо использовать алгоритм обучения многослойной ИНС с адаптивным шагом.

**Код программы:**

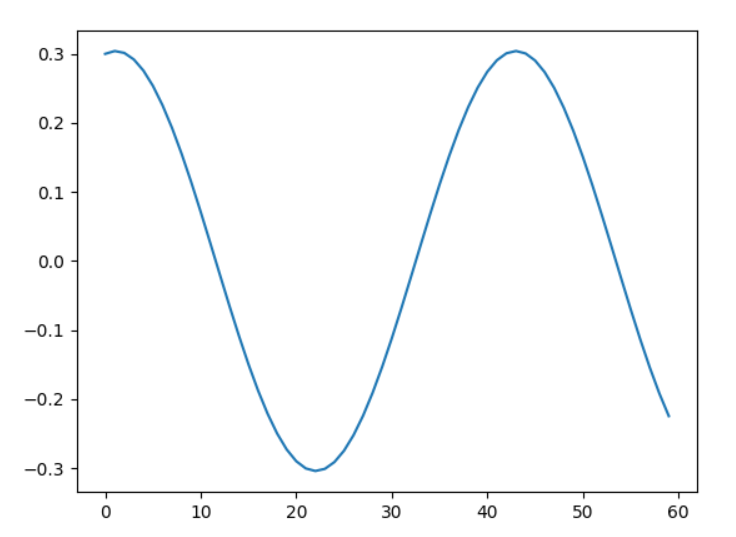
import random  
import math  
from prettytable import PrettyTable  
import matplotlib.pyplot as plt  
  
mistake\_min = 1e-7  
a, b, c, d = 0.3, 0.5, 0.05, 0.5  
numin\_nn = 8 # количество входов инс  
numin\_nel\_hidlayer = 3 # количество нэ в скрытом слое  
w\_ij = [[random.uniform(-0.1, 0.1) for i in range(numin\_nel\_hidlayer)] for j in range(numin\_nn - 1)]  
w\_jk = [random.uniform(-0.1, 0.1) for \_ in range(numin\_nel\_hidlayer)]  
thresholdValue\_j = [random.uniform(-0.5, 0.5) for \_ in range(numin\_nel\_hidlayer)]  
thresholdValue\_k = random.uniform(-0.5, 0.5)  
step\_j, step\_k = 1, 1  
#thresholdValue\_j = [0] \* numin\_nel\_hidlayer  
#thresholdValue\_k = 0  
  
  
def inputElements(ind):  
 y = []  
 new\_y = []  
 for i in range(numin\_nn):  
 if i == numin\_nn - 1:  
 y.append(new\_y)  
 y.append(a \* math.cos(b \* ind \* 0.3) + c \* math.sin(d \* ind \* 0.3))  
 else:  
 new\_y.append(a \* math.cos(b \* ind \* 0.3) + c \* math.sin(d \* ind \* 0.3))  
 ind += 1  
 return (y)  
  
  
def hiddenLayer\_Sj(j, y):  
 S\_j = 0  
 for i in range(numin\_nn - 1):  
 S\_j += y[0][i] \* w\_ij[i][j]  
 S\_j -= thresholdValue\_j[j]  
 return S\_j  
  
  
def sigmoidActivationFunction(j, y):  
 S\_j = hiddenLayer\_Sj(j, y)  
 return (1 / (1 + math.exp(-1 \* S\_j)))  
  
  
def outputLayer\_Sk(y\_j):  
 S\_k = 0  
 for i in range(numin\_nel\_hidlayer):  
 S\_k += y\_j[i] \* w\_jk[i]  
 S\_k -= thresholdValue\_k  
 return S\_k  
  
  
def change\_w\_jk(y\_j, mistake, step\_k):  
 for i in range(numin\_nel\_hidlayer):  
 w\_jk[i] = w\_jk[i] - step\_k \* mistake \* y\_j[i]  
  
  
def change\_w\_ij(y\_j, mistake, y, step\_j):  
 for i in range(numin\_nn - 1):  
 for j in range(numin\_nel\_hidlayer):  
 w\_ij[i][j] = w\_ij[i][j] -step\_j \* mistake \* w\_jk[j] \* y[0][i] \* (y\_j[j] \* (1 - y\_j[j]))  
  
  
def change\_thresholdValue\_j(y\_j, mistake, step\_j):  
 for i in range(numin\_nel\_hidlayer):  
 thresholdValue\_j[i] += step\_j \* mistake \* y\_j[i] \* (1 - y\_j[i])  
  
  
def new\_step\_sigmoidActivationFunction(y\_j, y\_k, mistake):  
 numerator, divider, divider\_2 = 0, 0, 0  
 mistake\_j = []  
 for i in range(numin\_nel\_hidlayer):  
 mistake\_j.append( mistake \* y\_k \* (1 - y\_k) \* w\_jk[i])  
 for i in range(numin\_nel\_hidlayer):  
 numerator += (mistake\_j[i]\*\*2) \* y\_j[i]\* (1 - y\_j[i])  
 divider += (mistake\_j[i]\*\*2) \* (y\_j[i]\*\*2)\* ((1 - y\_j[i])\*\*2)  
 step\_j = (4 \* numerator) / ((1 + y\_k\*\*2) \* divider)  
 return step\_j  
  
  
  
  
def main():  
 global thresholdValue\_k  
 global step\_k  
 global step\_j  
  
  
 #print("w\_jk: ", w\_jk)  
 #print("w\_ij: ", w\_ij)  
 #print("thresholdValue\_j: ", thresholdValue\_j)  
 #print("thresholdValue\_k: ", thresholdValue\_k)  
 count = 0  
  
 while True:  
 for k in range(30):  
 y\_j, y = [], []  
 y\_k, error = 0, 0  
 y = inputElements(k)  
 for i in range(numin\_nel\_hidlayer):  
 y\_j.append(sigmoidActivationFunction(i, y))  
 y\_k = outputLayer\_Sk(y\_j)  
 mistake = y\_k - y[-1]  
 error += 0.5 \* (mistake \*\* 2)  
 step\_j = new\_step\_sigmoidActivationFunction(y\_j, y\_k, mistake)  
 step\_k = 1 / (1 + (y\_j[0] \*\* 2) + (y\_j[1] \*\* 2) + (y\_j[2] \*\* 2))  
 change\_w\_jk(y\_j, mistake, step\_k)  
 change\_w\_ij(y\_j, mistake, y, step\_j)  
 change\_thresholdValue\_j(y\_j, mistake, step\_j)  
 thresholdValue\_k += mistake \* step\_k  
  
  
 #print("w\_jk: ", w\_jk)  
 #print("w\_ij: ", w\_ij)  
 #print("thresholdValue\_j: ", thresholdValue\_j)  
 #print("thresholdValue\_k: ", thresholdValue\_k)  
 if abs(error) < mistake\_min:  
 print("finaly: ", error)  
 break  
 print(count+1,";",error)  
 count += 1  
 print(count)  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()  
 mytable = PrettyTable()  
 mytable.field\_names = ["Right:", "Program:", "Error:"]  
 x, y = [], []  
 for i in range(30):  
  
 x.append(i)  
 test\_vector\_j = []  
 test\_vector = inputElements(i)  
 for j in range(numin\_nel\_hidlayer):  
 test\_vector\_j.append(sigmoidActivationFunction(j, test\_vector))  
 prog\_result = outputLayer\_Sk(test\_vector\_j)  
  
 right\_result = test\_vector[-1]  
 y.append(abs(prog\_result - right\_result))  
 mytable.add\_row([right\_result, prog\_result, abs(prog\_result - right\_result)])  
 plt.plot(x, y)  
 plt.show()  
 print(mytable)

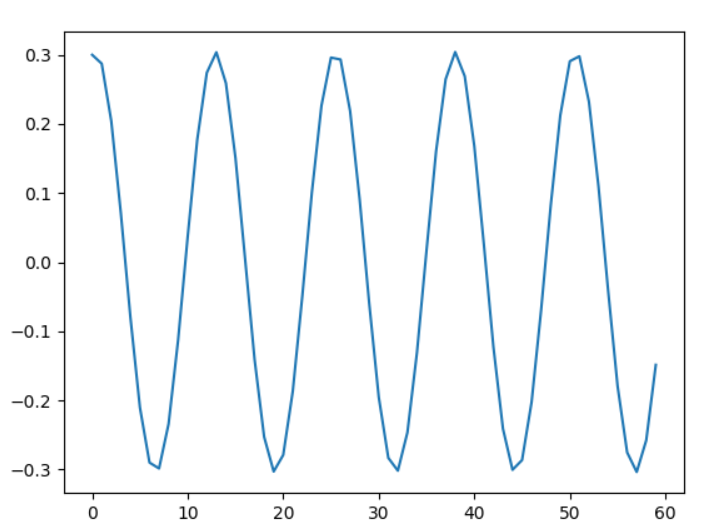
**Результат выполнения программы:**



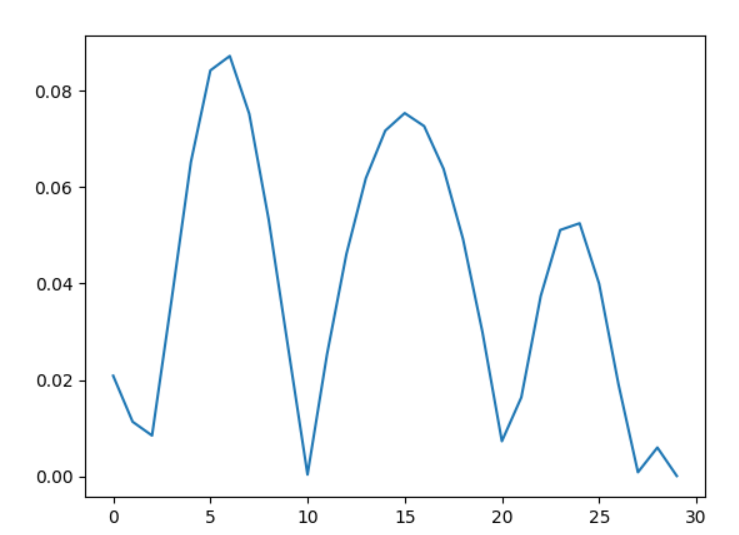


**График прогнозируемой функции на участке обучения:**





**График изменения ошибки в зависимости от итерации:**



**Вывод:** **:** изучила обучение и функционирование нелинейной ИНС при решении задач распознавания образов. Применив адаптивный шаг обучения, проанализировала, что обучения нс происходит гораздо быстрее.